Evaluating DHT-Based Service Placement for Stream Based Overlays

Evaluating DHT-Based Service Placement for Stream-Based Overlays

Peter Pietzuch, Jeffrey Shneidman, Jonathan Ledlie,
Matt Welsh, Margo Seltzer, Mema Roussopoulos

Harvard University

hourglass@eecs.harvard.edu

Abstract

Stream-based overlay networks (SBONs) are one approach to implementing large-scale stream processing systems. A fundamental consideration in an SBON is that of *service placement*, which determines the physical location of in-network processing services or operators, in such a way that network resources are used efficiently. Service placement consists of two components: *node discovery*, which selects a candidate set of nodes on which services might be placed, and *node selection*, which chooses the particular node to host a service. By viewing the placement problem as the composition of these two processes we can trade-off quality and efficiency between them. A bad discovery scheme can yield a good placement, but at the cost of an expensive selection mechanism.

Recent work on operator placement [3, 9] proposes to leverage routing paths in a distributed hash table (DHT) to obtain a set of candidate nodes for service placement. We evaluate the appropriateness of using DHT routing paths for service placement in an SBON, when aiming to minimize network usage. For this, we consider two DHT-based algorithms for node discovery, which use either the *union* or *intersection* of DHT routing paths in the SBON, and compare their performance to other techniques. We show that current DHT-based schemes are actually rather poor node discovery algorithms, when minimizing network utilization. An efficient DHT may not traverse enough hops to obtain a sufficiently large candidate set for placement. The union of DHT routes may result in a low-quality set of discovered nodes that requires an expensive node selection algorithm. Finally, the intersection of DHT routes relies on route convergence, which prevents the placement of services with a large fan-in.

1 Introduction

A marriage between the database and networking communities has produced a series of interesting systems for continous queries, large-scale stream processing, and application-level multicast. These systems are examples of a generic class of *stream-based overlay networks* (SBONs). SBON applications include real-time processing of financial data streams (Aurora [2], Borealis [1]), Internet health monitoring (PIER [9]) and querying geographically diverse sensor networks (IrisNet [8]).

SBONs pose two important challenges. First, a suitable choice of services, such as database operators, multicast points, or stream processors, must be provided by the system to satisfy user requirements. Second, these services must be deployed efficiently in the network according to user queries. Thus far, most existing research into SBONs has focused on the former question, with much less emphasis on efficient service placement. However, network-aware service placement becomes a crucial factor that determines the scalability and impact of an SBON when deployed on a shared networking infrastructure. Therefore, a service placement algorithm should be scalable and adaptive, and should perform well based on several cost metrics, such as network utilization and application latency.

Service placement is actually composed of two mechanisms: *node discovery* and *node selection*. Discovery is the process of identifying a set of nodes capable of hosting a service; we call this set of nodes the *candidate set*. Selection is the act of selecting a particular member of the candidate set to actually host the service. Traditionally, these two mechanisms have been intertwined, but by viewing them as separable processes, it is possible to gain greater insight into the performance of existing systems and develop new approaches to placing services.

In this paper, we investigate how well-suited current DHTs are to the task of node discovery with respect to efficient network utilization. We evaluate two DHT-based placement algorithms in comparison to non-DHT-based approaches, such as a globally optimal placement algorithm and a scheme based on spring relaxation [11]. Our analysis highlights the tight relationship between discovery and placement. A bad discovery mechanism can sometimes yield a good placement, but at the cost of an expensive selection mechanism. For the topologies we have considered, DHT-based schemes produce candidate sets that are marginally distinguishable from a random sampling. In particular, the union of DHT paths from producers to consumers creates a large collection of nodes and selecting the best one does yield a good placement, but we would have done equally well by selecting nodes at random. When considering the intersection of routing paths, services with a large fan-in are always placed at consumer nodes.

We conclude that current DHTs are not well-suited to this particular challenge of optimizing network utilization. We suggest that one should turn toward alternate solutions, such as the relaxation-based approach analyzed here, or a new generation of DHTs that are designed to address the needs of SBONs.

The outline of paper is as follows. Section 2 summarizes SBONs and describes the service placement problem. Section 3 introduces several node discovery and selection schemes that are then evaluated in Section 4. In Section 5 we review related work and Section 6 concludes.

2 Stream-based Overlay Networks

An SBON is an overlay network that streams data from one or more producers to one or more consumers, possibly via one or more operator services that perform in-network pro-

cessing. In an SBON, *circuits* interconnect multiple *services*. A circuit is a tree that specifies the identities and relationships between services in a data stream and corresponds to a query. Services that are part of a circuit are connected with *circuit links*.

We model a circuit as a logical query statement that is then *realized* on physical nodes. Some logical elements are constrained when the query is first stated. For example, the destination and data sources are specific physical nodes. We call these elements *consumer* and *producer* services, respectively, and consider them *pinned* because their logical-to-physical mapping is fixed. Other services, *e.g.*, a *join* operator, might be placed at any appropriate node in the network. We call these unassigned logical services *unpinned*. Logically, a join operator resides between two or more producers and one or more consumers, but its physical mapping is unassigned, *i.e.*, it is initially *unplaced*.

2.1 Placement Problem

Determining a placement for unpinned services is the fundamental placement problem in an SBON. Some placements are better than others: each placement has a *cost* and the quality of a placement is revealed by a *cost function*. Therefore, a solution to the placement problem calculates a valid placement for all unplaced services that minimizes the total incurred cost in the SBON.

Cost functions in an SBON can be categorized into two classes. Minimizing *application-specific* costs, such as circuit delay and jitter, addresses the application's desire for quality of service in the SBON. *Global* cost functions, such as network utilization and resource contention, attempt to capture the impact of a placement decision on other participants of the SBON.

In this paper, we concentrate on the global cost of utilizing the network when streaming data through the SBON, which is important in cooperative network environment, such as PlanetLab. One way to capture overall network utilization is the *bandwidth-latency* (BW-Lat) product, which is the sum of the *data rates* consumed by circuit links multiplied by their communication *latencies* calculated over all circuit links. The BW-Lat product captures network utilization as the amount of in-transit data in the network at a particular point in time.

The rationale behind this cost function is that the less data is put into the network by a placed circuit, the more network capacity is available to other circuits or applications. The BW-Lat cost function makes the assumption that high latency network links are more costly to use than low latency ones. Often high latency indicates network congestion or long geographical distance that means higher network operating costs. In both cases, the utilization of such links should be reduced. By factoring in the used bandwidth of a circuit link into the BW-Lat metric, the cost is proportional to the amount of network traffic used by a circuit. In other words, overall network utilization can be reduced more when good placement decisions are chosen for circuits with a high data rate.

3 Placement Algorithms

Many service placement algorithms can be viewed as consisting of two steps: *node discovery* and *node selection*. *Node discovery* identifies a subset of all nodes in the SBON as a possible candidate set for service placement, and *node selection* chooses a suitable node for the actual placement. An optimal node selection would consider all nodes in the SBON, but requiring global knowledge is clearly not feasible for a scalable system. Even in a moderately-sized network, such as PlanetLab, up-to-date node characteristics for 500 nodes cannot be gathered in a resource efficient manner. Therefore, most placement algorithms use the results of a node discovery scheme as the input for node selection to cope with the complexity of the placement problem. Other placement algorithms, such as the Relaxation placement scheme described below, reverse the ordering of the two steps or coalesce them into one.

3.1 Node Discovery

The goal of *node discovery* is to generate a list of physical nodes on which an unpinned service can be placed. This list is known as the *candidate set*. The quality of the candidate set, in terms of the placement cost, is an important consideration: if no nodes with a low placement cost are part of the candidate set, a good placement cannot be found even with an optimal node selection algorithm. The size of the candidate and the distribution of placement costs for the included nodes determines the amount of flexibility that the node selection algorithm has when the best choice from the set cannot support the placement due to resource limitations. In this section, we describe several possible candidate sets.

All. Setting the candidate set to be the entire overlay network gives the node selection algorithm the most flexibility to make a good placement. However, it is infeasible to maintain global knowledge about all nodes in a large-scale distributed system and process a large set of candidate nodes efficiently.

Consumer. This algorithm returns the node hosting the consumer service as the placement location, which models a centralized data warehouse system. While it trivially solves the placement problem, it makes no attempt to optimize the placement decision.

Producer. Since data producers are pinned services in the circuit, one can select these nodes as the candidate set. Using known producer nodes solves the discovery problem, but can result in a small, badly-chosen candidate set.

Random. A candidate set of k random nodes can be discovered through some mechanism. However, the average quality of this set may be worse than that of any other scheme that favors nodes with lower placement costs.

DHT Routing Path. A natural way to build a candidate set is to route a message between pinned services through an overlay network, such as a DHT. In a DHT setting, a message will traverse $\lceil \log_{\mathrm{b}}(N) \rceil$ hops in the worst case, where N is the number of nodes in the DHT and b is the numeric base used for hash keys during routing. There are two obvious ways to generate a candidate set when a circuit contains a consumer and multiple producers:

1. **DHT Union** takes the total set of overlay nodes in the paths from producers to the consumer as the candidate set. The service is then placed at one of these nodes.

2. **DHT Intersection** takes the intersection of overlay nodes in the routing path from producers to the consumer. The service is then placed at one of these ordered nodes, such as the node closest to the producers.

The goal of this paper is to explore the performance of these two DHT routing schemes in comparison with the other schemes and to determine their applicability for service placement in an SBON.

3.2 Node Selection

For each unplaced service in a circuit, the *node selection* algorithm must place the service on some node in the candidate set. In this paper, we consider three general selection algorithms. Other selection schemes are possible, but will produce placements no better than optimal selection. Of course, no selection algorithm is necessary when the candidate set contains only a single node.

Random selects a node out of the candidate set with uniform probability. This is a trivial scheme to implement, and may do well with a well-chosen candidate set. However, it does not attempt to optimize the placement decision.

Optimal chooses the best node from the candidate set with respect to some metric, such as network utilization or application latency. In this paper, we consider the BW-Lat product from Section 2.1. If the candidate set is well-chosen or large, optimal may find a globally-optimal placement. However, an efficient implementation of optimal selection is hard. An exhaustive search over all possibilities may result in a large amount of network probing and computational overhead for non-trivial circuits.

Relaxation [11] places services using a spring-relaxation model in an artificial coordinate space [7], in which distance corresponds to latency between physical nodes in the SBON. The placement coordinate is then mapped back to physical network space to perform the actual service placement. Prior work has shown that relaxation placement performs well compared to other algorithms and supports scalable and dynamic cross-circuit optimization decision [11]. However, relaxation requires additional overhead in calculating the latency space in a distributed fashion and maintaining a mapping back to physical space.

4 Evaluation

In this section we present our evaluation of DHT-based service placement compared to other, non-DHT-based placement schemes. The goal is to determine the performance of the DHTUnion and DHTIntersection algorithms when applied to different topologies and DHT parameters. Our evaluation focuses on the efficiency of network utilization, as captured by the BW-Lat product. Throughout this section, we refer to a placement algorithm by its discovery and selection schemes, *e.g.*, *All/Random*. The Optimal selection scheme uses the BW-Lat product as its metric.

4.1 Experimental Set-up

To evaluate the placement efficiency for a large number of circuits, we implemented a discrete-event simulator that operates either on the PlanetLab [16] topology with 186 nodes generated from all-pairs-ping measurements [15], or an artificial 600-node transit-stub topology created by the GATech topology generator [17]. After placing 1000 circuits each consisting of 4 pinned producers, 1 unpinned service, and 1 pinned consumer, the simulator calculates the placement cost per circuit for each of the placement algorithms. The four producers in the

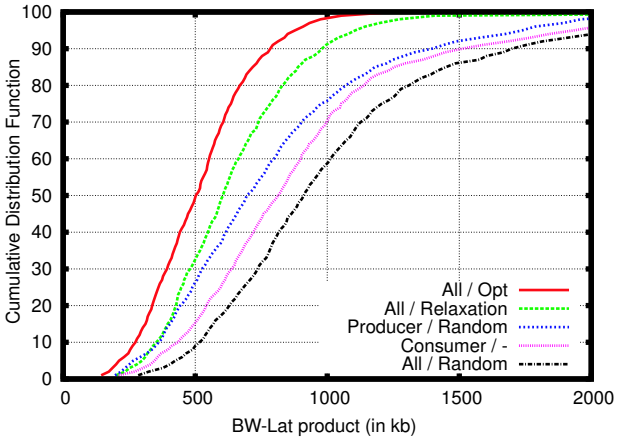

Figure 1: **Non-DHT**: CDF of the BW-Lat product on the PlanetLab topology.

Node		Topology			
Discovery	**Selection**	**PlanetLab**		**Transit-Stub**	
		b=2	**b**=64	**b**=2	**b**=64
All	Optimal	1.00		1.00	
All	Relax.	1.24		1.09	
RandomSet(6)	Optimal	1.26		1.36	
Producer	Random	1.60		1.43	
Consumer	—	1.70		1.63	
All	Random	1.96		1.84	
DHTUnion	Optimal	1.13	1.13	1.14	1.18
DHTUnion-NoProd	Optimal	1.17	1.31	1.16	1.31
DHTUnion	Random	1.60	1.63	1.56	1.52
DHTIntersec.	—	1.68	1.67	1.63	1.63
DHTIntersec.-Split	—	1.61	1.55	1.59	1.54
DHTIntersec.-Data	—	2.82	1.96	3.31	2.25

Table 1: 80th percentile of the BW-Lat product as a ratio of the 80th percentile of *All/Opt* after placing 1000 circuits for two topologies with DHT bases 2 and 64.

circuits produce streams with a data rate of $2\,\mathrm{kb/s}$ each, which are then aggregated into a single $1\,\mathrm{kb/s}$ stream by the unpinned service.

Two separate DHT implementations were used for the DHT-based placement schemes. We leverage a recent DHT implementation by crawling the *OpenHash* [10] routing tables running on PlanetLab, which uses the *Bamboo* [12] routing algorithm. OpenHash has a DHT key base of 2 and a leaf set size of 8. We performed latency measurement with *Scriptroute* [14] to fill in the missing Bamboo nodes in the all-pair pings data. We also implemented our own Pastry-like DHT, called *Pan*, which allowed us to vary the key base. A comparison of Pan and Bamboo routing with key base 2 shows that the average routing hop count of Pan is within 2 % of Bamboo's value. Both DHTs used are proximity-aware because otherwise the DHT routing paths would be essentially random. The Pan implementation follows Pastry's approach to achieve proximity awareness by prefering DHT nodes for its routing tables that are close in terms of latency.

4.2 Network Utilization

The experiment in Figure 1 depicts the efficiency of network utilization in terms of the amount of data in the

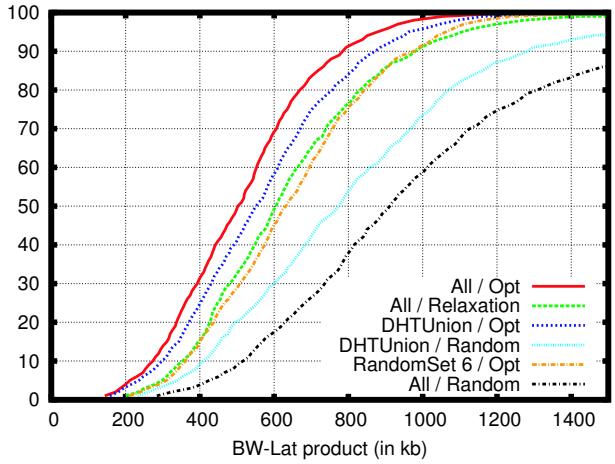

Figure 2: **DHTUnion**: CDF of the BW-Lat product with Bamboo (base=2) on the PlanetLab topology.

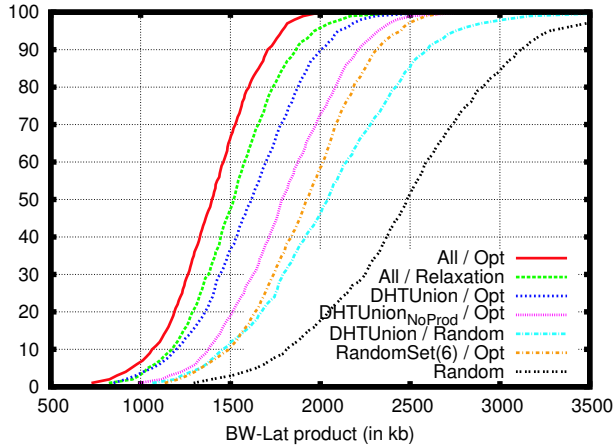

Figure 3: **DHT-Union**: CDF of the BW-Lat product with Pan (base=64) on 600-node transit-stub topology.

network for five different, non-DHT placement schemes. Each curve shows the BW-Lat distribution as a CDF after placing 1000 circuits. As expected, *All/Opt* performs best and *All/Random* worst. *All/Relaxation* is close to optimal, and outperforms the random selection of a producer (*Producer/Random*) and consumer placement (*Consumer/—*). We will use these placement schemes as baselines for comparison to DHT-based placement. All our experimental results for DHT-based service placement are summarized in Table 1. The data is listed as the ratio of the 80th percentile of the BW-Lat product compared to the 80th percentile of *All/Opt* after placing 1000 circuits using various placement schemes. In the next two sections, we discuss the results for two DHT-based node discovery schemes, *DHTUnion* and *DHTIntersection*, using several topologies and DHT parameters.

4.2.1 DHTUnion

The DHTUnion scheme for node selection uses the DHT routing paths from the producers to the consumer in a circuit to obtain a set of candidate nodes for service placement. The size and the quality of the set with respect to the placement cost function, will depend on the network topology and the specifics of the particular DHT, such as its network awareness, key base and leaf set size. Most DHTs are optimized for efficient key retrieval, which means that the number of routing hops is kept low by choosing a large key base. However, this reduces the size of the candidate set for node selection when using a DHT, potentially missing good placement nodes from the set. In terms of quality, the choice of the routing path by the DHT will determine the suitability of the candidate set for service placement.

PlanetLab Topology. In Figure 2, we plot the distribution of the BW-Lat product for three variations of the DHTUnion scheme on the PlanetLab topology using the Bamboo DHT. For such a small topology, many nodes are included in the candidate set because the Bamboo deployment on PlanetLab has a large average routing path length of 3.18 hops due to its binary key base. Therefore, the figure shows that *DHTUnion/Opt* placement performs well compared to *All/Opt*: the candidate set covers a significant fraction of all nodes and therefore is likely to include at least one good placement node. However, this good placement node must be found through an expensive exhaustive search.

The DHT contributes little to placement efficiency, which is supported by the fact that *RandomSet(6)/Opt* (1.26) performs similarly to *DHTUnion/Opt* (1.13). RandomSet uses a node size of six because this is close to the average number of nodes in the DHTUnion candidate set. A random choice of six nodes out of 186 is likely to include a good placement candidate. This is especially the case for the PlanetLab network, which mainly interlinks well-provisioned educational institutions [4]. In general, performing optimal node selection on large candidate sets is not desirable because of the probing and computational overheads when placing complex circuits with multiple unpinned services. The *DHTUnion/Random* algorithm (1.60) has a similar cost as *Producer/Random* (1.60) and *Consumer/—* (1.70) because of the probability that either the producer or consumer nodes are chosen randomly.

We study the effect of a more efficient DHT deployment on PlanetLab by simulating a DHT with a larger key base of 64. For this DHT, the average routing path length drops to 1.6 hops. Table 1 shows that the performance of DHTUnion/Opt is still good (1.13) when compared to *All/Opt*. Although the DHT routing paths are shorter due to the larger key base, the candidate set now becomes dominated by the 5 nodes hosting either pinned producers or consumers. We verify this claim with the DHTUnion-NoProd scheme: when the producer nodes are removed from the candidate set in *DHTUnion-NoProd/Opt* placement, the placement cost increases to 1.31. This means that the in-network DHT routing path is not long enough to contribute a good placement node.

Transit-Stub Topology. The problem of a small, low-quality candidate set, as returned by DHTUnion, is even more pronounced in larger topologies. In Figure 3, we consider an efficient DHT deployment with a key base of 64 deployed on a 600-node transit-stub topology. The average DHT path length here is 1.89 hops. In this topology, *DHTUnion/Opt* (1.18) performs worse than *All/Relaxation* (1.09). Removing the producer nodes (*DHTUnion-NoProd/Opt*) reduces the efficiency to 1.31, resulting in only a small gain when compared to *RandomSet(6)/Opt* (1.36).

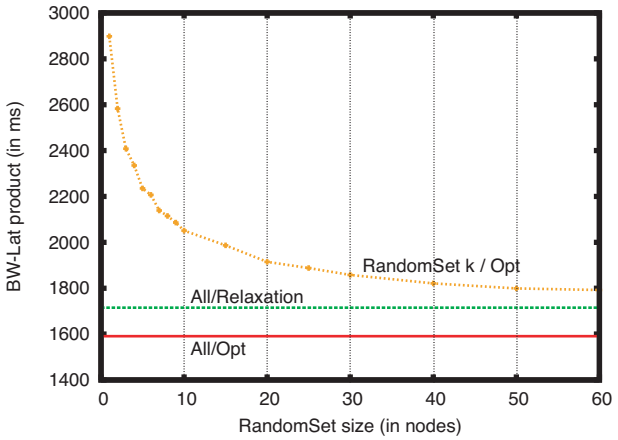

Figure 4: **RandomSet/Opt**: BW-Lat product with Pan (base=64) on 600-node transit-stub topology.

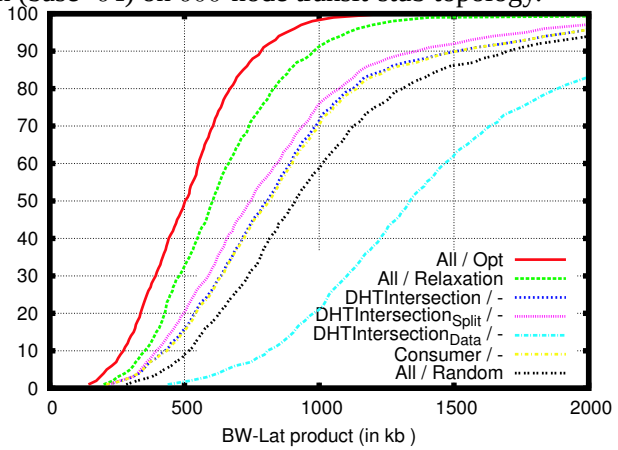

Figure 5: **DHT-Intersection**: CDF of the BW-Lat product with Bamboo (base=2) on the PlanetLab topology.

An obvious way to enlarge the candidate set is to consider the round-trip DHT routing path. However, increasing the size of candidate set without picking good nodes does not lead to efficient service placement in a large topology. The experiment in Figure 4 with the same 600-node transit-stub topology investigates the minimum size of a randomly chosen candidate set (RandomSet(k)) that is necessary to perform a good node selection with an exhaustive search (Opt). Even when the candidate set includes 10 % of all nodes, it does not achieve the performance of *All/Relaxation* placement. Note that RandomSet performs worse on the transit-stub topology than before because this network is not as homogeneous as PlanetLab.

4.2.2 DHTIntersection

The DHTIntersection discovery scheme considers the nodes in the intersection of all DHT routing paths between the producers and the consumer as the candidate set. It relies on the route convergence property of DHT routing [5], which states that routing paths to the same destination are likely to converge. A service is then placed on the farthest node along the path that is shared by all other routing paths. In this case, node discovery is equivalent to selection, so the selection scheme is represented by —.

In Figure 5, we evaluate three types of DHTIntersection. The overlapping curves of *DHTIntersection/*— and *Consumer/*— reveal that DHTIntersection performs only marginally better than consumer selection. In most cases the intersection of the four routing paths from the producers to the consumer contains only the node hosting the consumer. This means that DHTIntersection is not a good scheme for unpinned services with a large fan-in. Since the candidate set returned by DHTIntersection contains a single node in most cases, varying the node selection scheme is unnecessary.

The DHTIntersection-Split scheme recognizes this lack of convergence of multiple routes by assuming that an unpinned service can be *split* into sub-services. These sub-services can then be placed independently. A sub-service is created at the intersection between any two routing paths from producers to consumer. This is similar to the setup of multicast trees in DHT-based systems, such as Scribe [5], where a multicast node is created at the node joining the path from a new subscriber to the root of the multicast tree. The graph shows that splitting services improves the fraction of cases, in which services are placed in-network. However, it does not reach the performance of *All/Relaxation*, which optimizes for the BW-Lat product. Moreover, it is not applicable if services cannot be decomposed into sub-services.

Application-level multicast schemes based on DHTs often send data along DHT routing hops. This has the advantage that the data flows benefit from the resilience and load-balancing properties of the DHT but it also incurs the penalty of more network traffic in the system. To evaluate this effect, the final placement scheme is DHTIntersection-Data, as shown in Figure 5, which in addition to using the DHT for service placement, also routes the data itself through the DHT. The last row in Table 1 suggests that the penalty is directly related to the key base, which determines the number of hops in the routing path. For the Bamboo DHT on PlanetLab, the penalty of routing data through the DHT compared to *All/Opt* placement is almost a factor of 3. We suggest that many applications would benefit from including their own resilience mechanisms at the application-level without paying the price for using the DHT for data routing.

4.3 Summary

Our experiments suggest that DHTs are less efficient than non-DHT alternatives, such as Relaxation placement, for service placement in SBONs. DHTs are designed for efficient key lookup, but this yields small candidate set sizes for service placement. Using a small key base increases the number of routing hops thus helping node discovery but also reduces the efficiency of key lookup for other applications sharing the DHT. A large key base, such as in one-hop routing, may not return enough useful nodes for service placement.

Another drawback is the low quality of the obtained candidate set from the DHT, which makes it necessary to perform a costly exhaustive search through all placement possibilities. Fundamentally, DHTs are not designed to optimize stream-based processing in the overlay network. A topology-aware DHT uses only latency to optimize its overlay routing paths. Since DHTs are connection-less by nature and thus unaware of the data streams in the SBON, they cannot easily optimize their routing tables for efficient service placement. Algorithms, such as Relaxation place-

Node		
Discovery	**Selection**	**SBON Examples**
All	Random	PIER[9]
All	Relaxation	Hourglass[13]
All	Other (Human Decision)	GATES[6]
Consumer	—	Typ. Warehouse
Con. & Prod.	Varies (Random, Heuristic)	Typ. CQ System
DHTUnion	Greedy Heuristic on Opt	Borealis, SAND[1, 3]
DHTIntersec.	—	Scribe[5], Bayeux[18]

Table 2: SBONs classified by their placement techniques.

ment [11], designed specifically to minimize the amount of traffic in the SBON do not suffer from the same restrictions.

The intersection of DHT routing paths avoids the issue of a large, low-quality candidate set but has the problem that routing paths between pinned services may not converge. This is especially the case for services with a large number of incoming circuit links. Splitting services into sub-services addresses this problem but is not applicable in general. Routing data along DHT hops, as done by certain application-level multicast schemes, carries a large efficiency penalty.

5 Related Work

We suspect that the optimization of service placement decision in SBONs will be a growing research area, as can be seen from the wide-range of related work.

New continuous query (CQ) work breathes life into questions that arose in distributed database optimization. Most related to our work is *SAND* [3] in the context of the stream-processing system *Borealis* [1]. In our terminology, they use DHTUnion to pick the candidate set, and a greedy heuristic on optimal for node selection. Their scheme can do no better than a *DHTUnion/Opt* placement.

The location of operators and corresponding relational tables in *PIER* [9], a distributed database system built on top of a DHT, is determined through random hash selection from all overlay nodes and explicitly recognizes the possible inefficiency in such a service placement scheme, and is essentialy an *All/Random* placement.

Overlay routing and application level multicast systems consider where to place multicast services to best optimize an expanding array of metrics. For example, *Scribe* [5] uses a DHT to produce a multicast tree to connect publishers to subscribers. In our terminology, Scribe uses DHTIntersection-Split as its discovery mechanism and a "farthest common ancestor" heuristic as its selection mechanism.

Grid users are recognizing the need to use in-network services to help process massive data streams. For instance, *GATES* [6] provides a way of introducing processing services but requires these services be pre-placed by a system administrator.

6 Conclusions

In this paper, we have shown that current DHTs do not produce a particularly good candidate set of nodes for service placement. DHT Union does not provide significant discovery value beyond that of selecting a random set of nodes from the overlay. If services cannot be split, DHT Intersection tends to reduce to Consumer placement, resulting in highly restricted and poor placement. If services can be split, we found that DHT Intersection still does not perform as well as a dedicated mechanism like Relaxation.

These results suggest a number of areas for further research. First, it remains an open question of whether it is possible to construct a DHT that is sufficiently network-aware such that it could be used to easily construct a good candidate set for node placement. How might we construct such a DHT? What does it mean for a DHT to be dynamically aware of network conditions? Second, should we declare that DHTs are not the correct abstraction on top of which to construct service placement algorithms? What alternative structures are possible? Third, is it necessary to globally optimize streaming applications? Do we believe that there will be sufficiently large amounts of streaming traffic to warrant building a system that does cross-circuit optimization instead of just local optimization? Answering these questions is crucial for successful deployment of stream-based applications.

References

[1] D. Abadi, Y. Ahmad, H. Balakrishnan, et al. The Design of the Borealis Stream Processing Engine. Technical Report CS-04-08, Brown University, July 2004.

[2] D. Abadi, D. Carney, U. Cetintemel, et al. Aurora: A New Model and Architecture for Data Stream Management. *VLDB*, Aug. 2003.

[3] Y. Ahmad and U. Çetintemel. Network-Aware Query Processing for Stream-based Applications. In *VLDB*, Aug. 2004.

[4] S. Banerjee, T. G. Griffin, and M. Pias. The Interdomain Connectivity of PlanetLab Nodes. In *Passive and Active Measurement Workshop*, Antibes Juan-les-Pins, France, Apr. 2004.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A Large-scale and Decentralized Application-level Multicast Infrastructure. *JSAC*, 20(8), Oct. 2002.

[6] L. Chen, K. Reddy, and G. Agrawal. GATES: A Grid-Based Middleware for Processing Distributed Data Streams. In *HPDC-13*, Honolulu, Hawaii, June 2004.

[7] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A Decentralized Network Coordinate System. In *Proc. of the ACM SIGCOMM'04 Conference*, Portland, OR, Aug. 2004.

[8] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4), Oct. 2003.

[9] R. Huebsch, J. M. Hellerstein, N. Lanham, et al. Querying the Internet with PIER. In *VLDB)*, Berlin, Germany, Sept. 2003.

[10] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Spurring Adoption of DHTs with OpenHash, a Public DHT Service. In *Proc. of IPTPS'04*, San Diego, CA, Feb. 2004.

[11] P. Pietzuch, J. Shneidman, M. Welsh, M. Seltzer, and M. Roussopoulos. Path Optimization in Stream-Based Overlay Networks. Technical Report TR-26-04, Harvard University, October 2004.

[12] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *USENIX '04*, Boston, MA, June 2004.

[13] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh. Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. Technical Report TR-21-04, Harvard University, Sept. 2004.

[14] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A Public Internet Measurement Facility. In *USITS'02*, Mar. 2003.

[15] J. Stribling. All-Pairs-Pings for PlanetLab. http://www.pdos.lcs.mit.edu/~strib/pl_app/, Sept. 2004.

[16] The Planetlab Consortium. http://www.planet-lab.org, 2004.

[17] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *Proc of IEEE Infocom'96*, volume 2, pages 594–602, San Francisco, CA, Mar. 1996.

[18] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *NOSSDAV*, June 2002.